**Нововведения ES6 в JS. let, const, стрелочные функции, this**

## Объявление переменных с помощью let и const

Переменная типа **let** может изменять свое значение

let name = "Jane Smith";

name = "Jane Miller";

Переменная типа **const** не может менять значение.

const yearOfBirth = 1980;

yearOfBirth = 1920; // Error Assignment to constant variable.

Переменной типа **const** нужно сразу установить значение (инициализировать).

**ВЫВОДЫ**:

* Вместо **var** используем **let** или **const**.
* По умолчанию используем **const**.

Если мы сразу видим что переменная будет изменяться тогда используем let. Если сразу этого не можем предвидеть, тогда используем const и в случае возникновения ошибки из-за изменения переменной - проверим где это происходит, и если так и нужно - тогда меняем определение переменной на let.

***Отличия между var и let***:

* Область видимости переменных **var** ограничивается либо **функцией**, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока. Для **let** область видимости - **блок**.

function examResult(passedTest) {

if (passedTest) {

**var** name = "Юрий";

**var** message = name + " Поздравляю! Вы успешно сдали экзамен!";

console.log(message); // +

}

console.log(message); // +

}

console.log(message); // Error: message is not defined

examResult(true); // +

function examResult(passedTest) {

if (passedTest) {

**let** name = "Юрий";

**let** message = name + " Поздравляю! Вы успешно сдали экзамен!";

console.log(message); // +

}

console.log(message); // Error: message is not defined

}

examResult(true);

* Если в блоке кода дважды объявить одну и ту же переменную **let**, будет ошибка. Используя **var**, можно переобъявлять переменную сколько угодно раз. Повторные var игнорируются
* Для **var** переменную можно использовать **до ее объявления**. Другими словами, переменные var считаются объявленными с самого начала исполнения функции (или скрипта) вне зависимости от того, в каком месте функции реально находятся их объявления. Это поведение называется «hoisting» (всплытие, поднятие), потому что все объявления переменных var «всплывают» в самый верх функции. Для **let** и **const** не работает **hoisting**.

age = 35; // Нет ошибки

var age;

function examResult(passedTest) {

if (passedTest) {

let name = "Юрий";

let message = name + " Поздравляю! Вы успешно сдали экзамен!";

}

console.log(message); // Uncaught ReferenceError: message is not defined

}

examResult(true);

Поскольку все объявления переменных var обрабатываются в начале функции, мы можем ссылаться на них в любом месте. Однако, переменные имеют значение **undefined** до строки с присвоением значения.

## Блоки и IIFE. Приватность.

Если вам нужна **приватность** для **let** и **const**, то можно использовать **block scope**

// **ES5** - Облоасть видимости IIFE

(function () {

var userName = "Джон До";

console.log(userName); // переменная видна внутри

})();

console.log(userName); // и не вилна снаружи

// **ES6** - Блочная область видимости {}

{

const userName = "Джон До";

console.log(userName); // переменная видна внутри

}

console.log(userName); // и не видна снаружи

## Шаблонные строки

// **ES5**

console.log("Привет! Я " + name + ", мне " + calcAge(yearOfBirth) + " и я " + profession + ".");

// **ES6**

console.log(`Привет! Я ${name}, мне ${calcAge(yearOfBirth)} и я ${profession}.`);

## Новые методы для работы со строками

const name = "John Doe";

console.log("name.startsWith('J')", name.startsWith('J')); // true

console.log("name.endsWith('xoe')", name.endsWith('xoe')); // false

console.log("name.includes('hn D')", name.includes('hn D')); // true

console.log(name.repeat(4)); // John DoeJohn DoeJohn DoeJohn Doe

## Стрелочные функции - Синтаксис

// **ES5**

var calcAgeES5 = function (yearOfBirth){

return 2020 - yearOfBirth;

}

console.log(calcAgeES5(1980));

// **ES6**

*Полный синтаксис () => {};*

var calcAgeES6 = (yearOfBirth) => {

return 2020 - yearOfBirth;

}

console.log(calcAgeES6(1980));

*Сокращенный синтаксис (если сразу идет возврат выражения) () => ;*

var calcAgeES6 = (yearOfBirth) => 2020 - yearOfBirth;

console.log(calcAgeES6(1980));

*Пример использования:*

// **ES5**

var agesES5 = years.map(function (item) {

return 2020 - item;

});

console.log(agesES5);

// **ES6**

const agesES6 = years.map((item) => 2020 - item);

console.log(agesES6);

Запись в виде стрелочной функции удобно использовать для определения **callback** функций

## Стрелочные функции и ключевое слово this

//  **ES5**

*var objectOne = {*

*objectName: "Object One",*

*sayHi: function () {*

*console.log(this); // {objectName: 'Object One', sayHi: ƒ}*

*console.log(this.objectName); // Object One*

*function printToConsole() {*

*console.log(this); //Window {window: Window, self: Window, document: document, name: '', location: Location, …}*

*console.log(this.objectName);     //undefined*

*}*

*printToConsole();*

*},*

*};*

*objectOne.sayHi();*

// **ES6**

*var objectOne = {*

*objectName: "Object One",*

*sayHi: function () {*

*console.log(this); //{objectName: 'Object One', sayHi: ƒ}*

*console.log(this.objectName); // Object One*

*const printToConsole = () => {*

*console.log(this);* *//{objectName: 'Object One', sayHi: ƒ}*

*console.log(this.objectName); // Object One*

*}*

*printToConsole();*

*}*

*}*

*objectOne.sayHi();*

Стрелочные функции особенные:

Внутри стрелочной функции **нет своего this**.

this в стрелочной функции - делит this с окружением первой нормальной родительской функции.

1. Идем вверх до первой нормальной функции (метода или объекта)

2. Смотрим в каком окружении находится эта функция (метод или объект)

3. this "ссылается" на это окружение

***Пример*:**

var objectOuter = {

objectName: "Outer object",

someAction: function () {

console.log(this); // objectOuter

var objectOne = {

objectName: "Object One",

sayHi: () => {

console.log(this); // objectOuter

console.log(this.objectName); // "Outer object"

},

};

objectOne.sayHi();

},

};

objectOuter.someAction();

***Пример потери this:***

Проблема. this внутри функции callback ссылается на window, а не на объект который нам нужен

var obj = {

    objName: "Object with array",

    numbers: [10, 15, 25, 05],

    value: 100,

    increaseArrayOnValue: function () {

***this***.numbers.forEach(function (item) {

            console.log(***this***); *// Window*

            console.log(item); *// 10*

            console.log(***this***.value); *// undefined*

            console.log(item + ***this***.value); *// NaN*

        })

    }

}

obj.increaseArrayOnValue();

**Варианты решения:**

1. Сохраняем this в переменную

2. Привязка контекста через bind()

var obj = {

objName: "Object with array",

numbers: [10, 15, 25, 05],

value: 100,

increaseArrayOnValue: function () {

this.numbers.forEach(

function (item) {

console.log(this); // {objName: 'Object with array', numbers: Array(4), value: 100, increaseArrayOnValue: ƒ}

console.log(item); // 10

console.log(this.value); // 100

console.log(item + this.value); // 110

}.**bind(this)**

);

},

};

obj.increaseArrayOnValue();

3. Используем стрелочную функцию

var obj = {

  objName: "Object with array",

  numbers: [10, 15, 25, 05],

  value: 100,

  increaseArrayOnValue: function () {

*this*.numbers.forEach**((item) =>** {

      console.log(*this*); *// obj*

      console.log(item); *// 15*

      console.log(*this*.value); *// 100*

      console.log(item + *this*.value); *// 115*

    });

  },

};

obj.increaseArrayOnValue();